Build Heterogenous System using Yocto Project

Sandeep Gundlupet Raju, AMD

About Me

- Yocto Project Software Architect at AMD(formerly Xilinx)
- Using Linux since 2006 and OpenEmbedded(poky linux) since 2010
- Contributor to Yocto Project (poky, amd xilinx meta layers, meta-ros, meta-virtualization, meta-jupyter), OE, U-boot, Linux and Lopper
- Co-Maintainer of meta-jupyter and amd xilinx yocto meta layers
- Email me: sandeep.gundlupet-raju@amd.com
- Connect me: https://www.linkedin.com/in/sandeep-gundlupet-raju-91343020/

Agenda

- Summary
- Heterogenous System
- System Device Tree (S-DT)
- Yocto Project
- Generating YP Config using S-DT
- What Next in Scarthgap

Summary

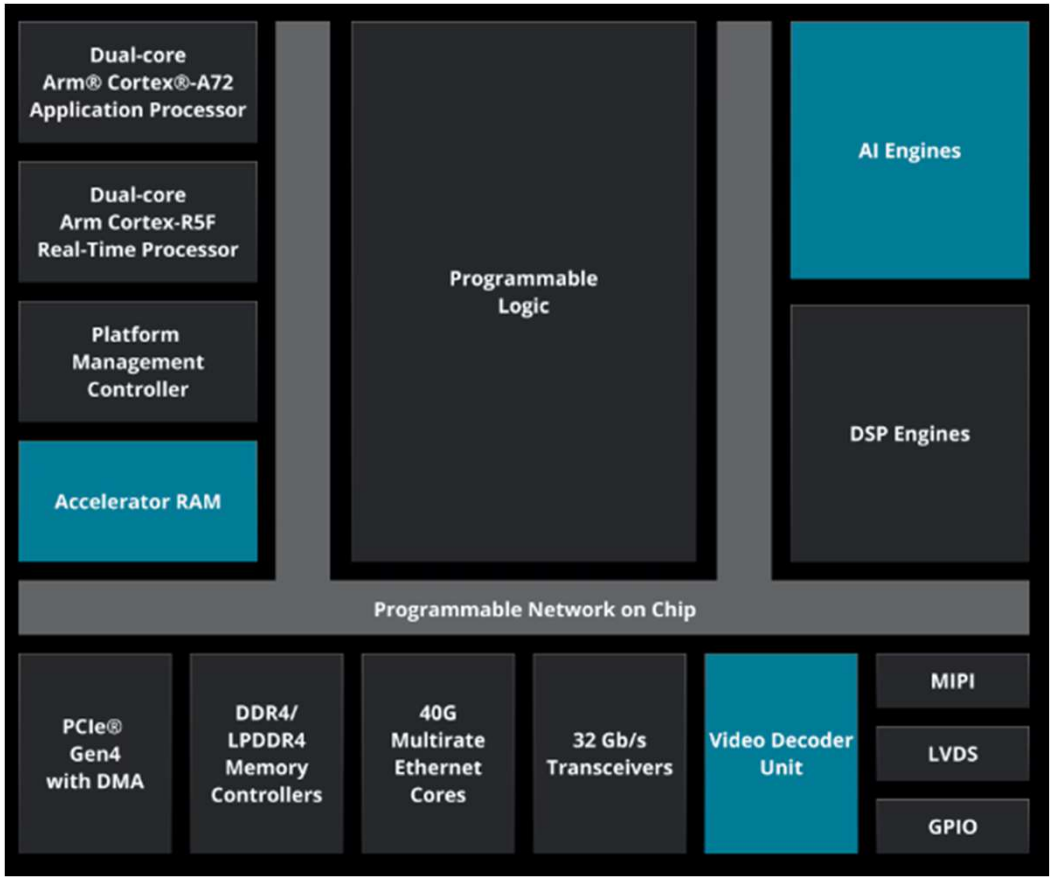
Summary

- Modern System on Chip(SoC) are convoluted, and designs requires an heterogenous systems includes Application Processing Unit(APU ex: A72's), Real Time Processing Unit(RPU ex: R5's) and Programmable Logic in a single SoC
- Need separate build configurations for multiple cpu clusters such as A72's R5's and MicroBlaze to run different OS or baremetal or RTOS in parallel on the same chip
- Software(Baremetal, RTOS and Linux OS) requires different hardware configuration such as memory allocation, address and register mapping etc, for different CPUs
- Device tree bindings are complex with standard shared memory configurations for asymmetric multiprocessing(AMP) systems which is done in an ad-hoc way

Heterogenous System

Heterogenous System

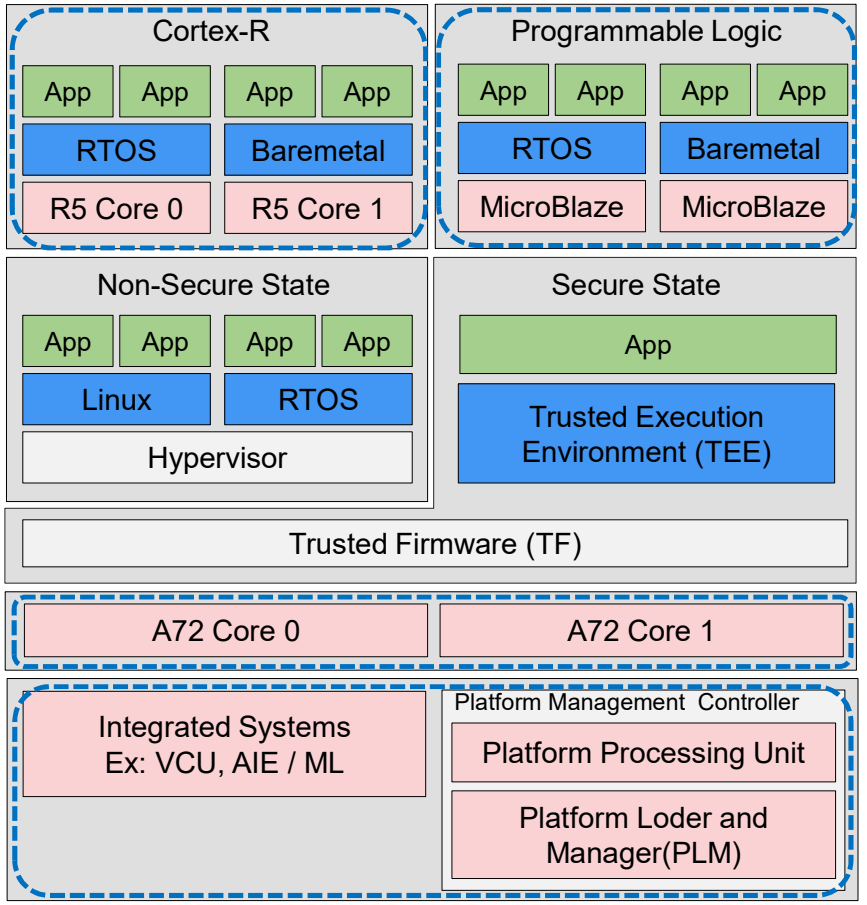
A system with multiple CPU's or clusters of CPU that uses shared memory and devices to be allocated. For example, a single SoC consists of APUs, RPUs, Programmable Logic, AI Engines, DSP Engines etc and each CPUs can run and execute different operating environments independently



AMD

Subsystems, Domains and Protections

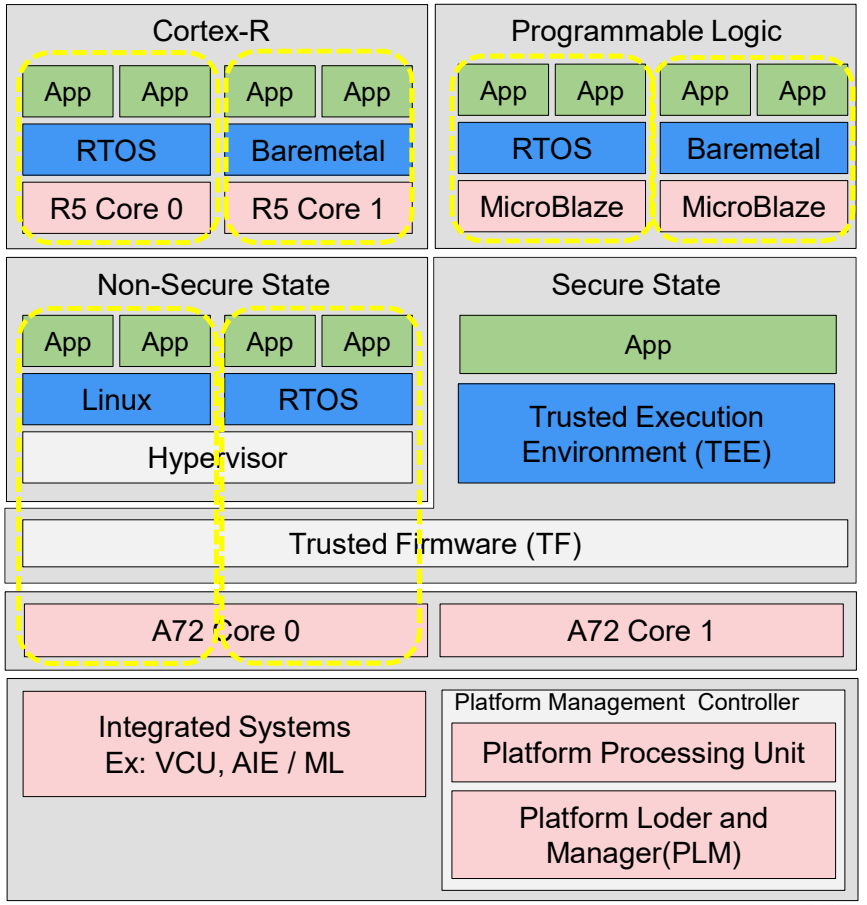
- A **Subsystem** is the outer grouping of hardware resources
- Contains zero or more Domains
- Can contain multiple CPU clusters
- A Subsystem can also be a Domain - Only one core or CPU cluster
- The subsystem info is used by the firmware
- Typically defined by HW Architect



AMD

Subsystems, Domains and Protections

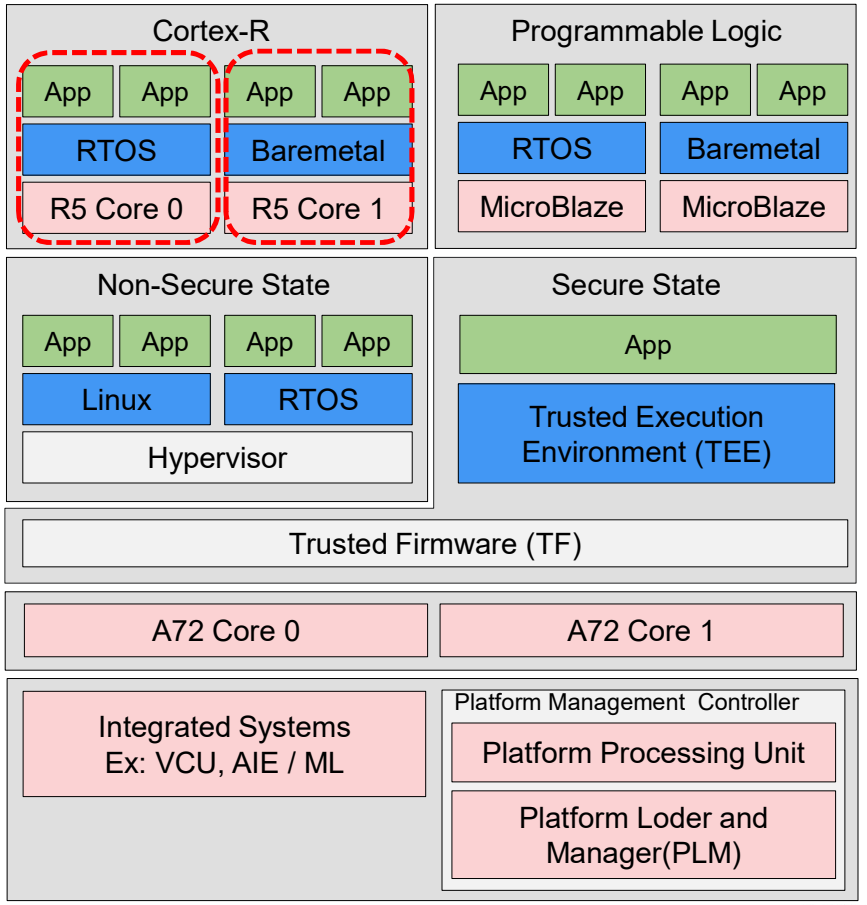
- A **Domain** is a grouping of HW resources used by SW
- Example an Baremetal or OS or Hypervisor
- Only one CPU or CPU cluster allowed
- Only one address space
- A Domain can contain other Domains
- Example a Hypervisor has a Domain for each VM
- The domain info is used by build/runtime of each OS/HV
- Typically defined by SW Architect



AMD

Subsystems, Domains and Protections

- **Protection** is used to protect a Subsystem or Domain
- Typically using HW registers
- Protect subordinates in domain from outside managers
- The protection info is used by the firmware
- Can be defined by HW and/or SW Architects
- Protection and lifecycle are independent



AMD

System Device Tree (S-DT)

Device Tree

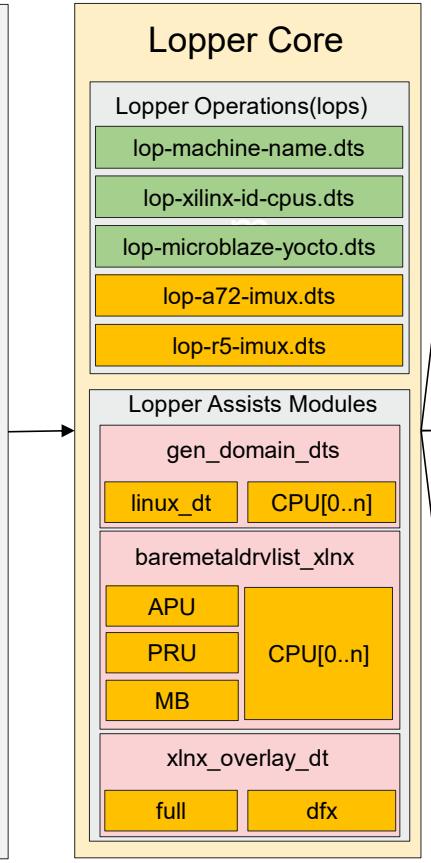
- Device tree is a concept which describe system hardware. A device tree is a tree data structure with nodes that describe the physical devices in a system such as CPU's, memory and peripheral devices
- Device tree are used in U-boot, Linux, Xen, Zephyr
- A boot program loads a device tree into a client program's memory and passes a pointer to the device tree to the client
- Traditional device trees are only describing the system with one address space
- An Heterogenous systems needs multiple device tree for each operating environment

System Device Tree(S-DT)

- An open-source device tree that defines a complete heterogenous system hardware
  - Domains – Set of nodes defining specific address space
  - Resources Group – Set of shared nodes grouped together
- It describes multiple CPU clusters and corresponding views of their address space
- An open-source tool called Lopper is used to transform device tree.
  - Prunes a system device tree to an operating environment(U-boot/Linux/Xen/Zephyr) standard device-tree
  - Lopper merges or removes nodes and resolves address mapping issues
- See https://static.linaro.org/connect/san19/presentations/san19-115.pdf for more details

S-DT Lopper Transformations

```
cpus_a72: cpus-a72@0 {
    compatible = "cpus,cluster";
    address-map = <0x0 0xf0000000 &amba 0x0 0xf0000000 0x0 0x10000000>,
    <0x0 0xf9000000 &amba_apu 0x0 0xf9000000 0x0 0x80000>,
    <0x0 0x00000000 &axi_noc_0_memory 0x0 0x00000000 0x0 0x80000000>,
    <0x00000008 0x80000000 &axi_noc_0_memory 0x00000008 0x80000000 0x0
0x80000000>,
    <0x0 0xf1010000 &ospi 0x0 0xf1010000 0x0 0x10000>,
    <0x0 0xf1020000 &gpio1 0x0 0xf1020000 0x0 0x10000>,
    <0x0 0xf1050000 &sdhci1 0x0 0xf1050000 0x0 0x10000>,
    [...]
    #ranges-address-cells = <0x2>;
    #ranges-size-cells = <0x2>;
};
cpus_r5_0: cpus-r5@0 {
    compatible = "cpus,cluster";
    address-map = <0xf0000000 &amba 0xf0000000 0x10000000>,
    <0xf9000000 &amba_rpu 0xf9000000 0x3000>,
    <0x40000 &axi_noc_0_memory 0x40000 0x7ffc0000>,
    <0xf1010000 &ospi 0xf1010000 0x10000>,
    <0xf1020000 &gpio1 0xf1020000 0x10000>,
    <0xf1050000 &sdhci1 0xf1050000 0x10000>,
    [...]
    #ranges-address-cells = <0x1>;
    #ranges-size-cells = <0x1>;
};
cpus_microblaze_0: cpus_microblaze@0 {
    compatible = "cpus,cluster";
    address-map = <0xf0000000 &amba 0xf0000000 0x10000000>,
    <0x00000000 &axi_noc_0_memory 0x00000000 0x80000000>,
    <0xff320000 &ipi_pmc 0xff320000 0x10000>,
    <0xff390000 &ipi_pmc_nobuf 0xff390000 0x10000>,
    <0xf1010000 &ospi 0xf1010000 0x10000>,
    <0xf1020000 &gpio1 0xf1020000 0x10000>,
    <0xf1050000 &sdhci1 0xf1050000 0x10000>,
    [...]
    #ranges-address-cells = <0x1>;
   #ranges-size-cells = <0x1>;
};
```




```
/dts-v1/;
/ {
    [...]
    psv_cortexa72_0: cpu@0 {
        compatible = "arm,cortex-a72", "arm,armv8";
        device_type = "cpu";
        enable-method = "psci";
        operating-points-v2 = <0x6f>;
        [...]
    };
    [...]
};
```

```
/dts-v1/;
/ {
    [...]
    psv_cortexr5_0: cpu@0 {
        compatible = "arm,cortex-r5";
        device_type = "cpu";
        reg = <0x0>;
        operating-points-v2 = <0x6f>;
      [...]
    };
    [...]
};
```

```
/dts-v1/;
/ {
    [...]
    psv_pmc_0: cpu@0 {
        compatible = "pmc-microblaze";
        device_type = "cpu";
        reg = <0x0>;
        operating-points-v2 = <0x6f>;
        [...]
    };
    [...]
};
```

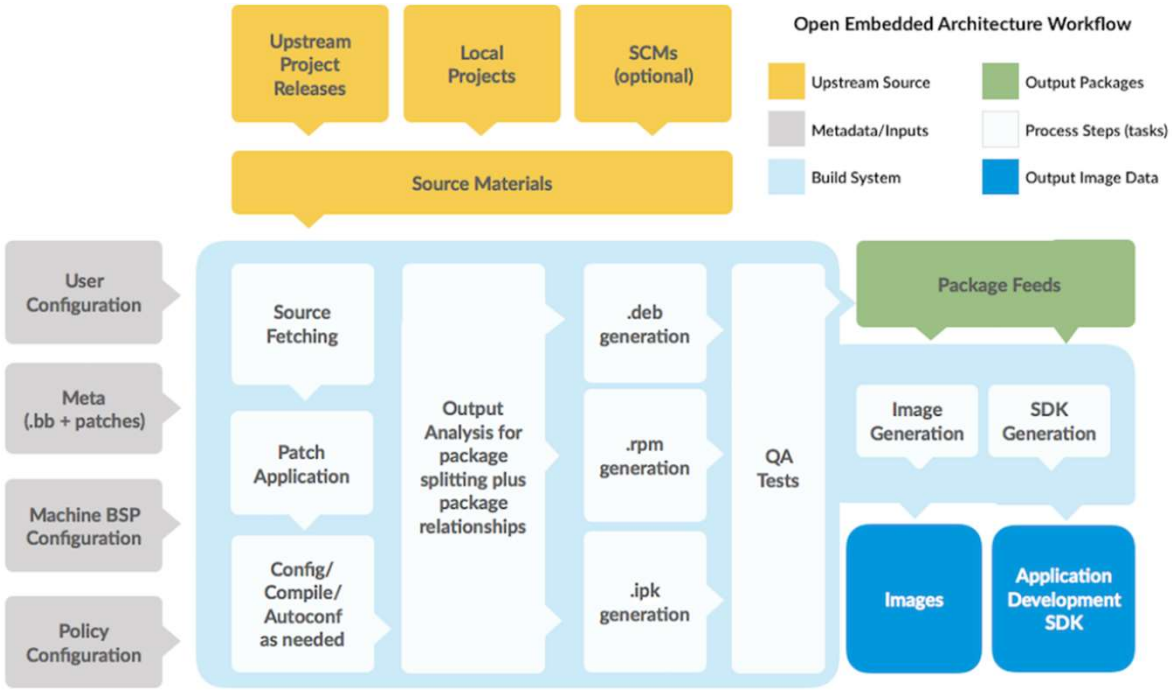
* lop dts highlighted in green is in meta-xilinx layer

AMD

Yocto Project

Yocto Project

- The Yocto Project is an open-source collaboration project that helps developers create custom Linux-based systems for embedded products and other targeted environments, regardless of the hardware architecture
- The project provides a flexible set of tools and configurations which can be used to create build Linux, baremetal, Zephyr, FreeRTOS, Hypervisor images for embedded devices



AMD

Yocto Project Overview

- **Bitbake** is a build engine(task executor and scheduler) of OpenEmbedded
- **Layer** is a collection of metadata(configuration files, recipes etc) organized into a file and directory structure
- **Recipe** is metadata file contains set of tasks on how to build a particular software package
- **Configuration files** contain global build system settings in the form of simple variable assignments
- **Machine** describes a hardware configuration files based on processor or SoC tuning files

See: https://docs.yoctoproject.org/dev/singleindex.html

User Configuration

- **local.conf** is where you override and define what you are building
  - BB_NUMBER_THREADS
  - PARALLEL_MAKE
  - MACHINE settings
  - DISTRO settings
  - EXTRA_IMAGE_FEATURES
- **bblayers.conf** is where you configure with layers to use
  - Add Yocto Project Compatible layers to the BBLAYERS
  - Default: meta (oe-core), meta-yocto and meta-yocto-bsp

```
build/
└── conf
    ├── bblayers.conf
    └── local.conf
```

Yocto Project Multiple Configuration(multiconfig) Builds

- You can use a single bitbake command to build multiple images or packages for different targets where each image or package requires a different configuration (multiple configuration builds)
- **BBMULTICONFIG** specifies each additional separate configuration when you are building targets with multiple configurations. Ex: `BBMULTICONFIG = "x86 arm"`

```
build/
└── conf
    ├── bblayers.conf
    ├── local.conf
    └── multiconfig
        ├── arm.conf
        └── x86.conf
```

Yocto Project Multiple Configuration (multiconfig) Builds

- Bitbake
  - `$ bitbake target`
  - `$ bitbake core-image-minimal`
- Build dependency
  - `task[depends] = "recipe_name:task_on_which_to_depend"`
  - `do_image[depends] = "core-image-minimal:do_rootfs"`

- Multiconfig Bitbake
  - `$ bitbake [mc:multiconfigname:]target [[[mc:multiconfigname:]target] ... ]`
  - `$ bitbake mc:x86:core-image-minimal mc:arm:core-image-sato mc::core-image-base`
- Multiconfig Build dependency
  - `task_or_package[mcdepends] = "mc:from_multiconfig:to_multiconfig:recipe_name:task_on_which_to_depend"`
  - `do_image[mcdepends] = "mc:x86:arm:core-image-minimal:do_rootfs"`
  - Note:
    - from_multiconfig can be blank
    - `bitbake mc:default:<target>` is the same as what you see on the left. i.e., `bitbake <target>`

See: https://docs.yoctoproject.org/dev/singleindex.html#building-images-for-multiple-targets-using-multiple-configurations

Generating YP Config using S-DT

Pre-requisites

- Download and Install AMD Vivado
- Use AMD Vivado tools create a Versal AI Edge design(synthesis -> implementation) and generate the output hardware description file (xsa)
- Convert xsa to S-DT using AMD system-device-tree generator by following the README instructions
- Clone poky, meta-openembedded, meta-virtualization and meta-xilinx layers
- Initialize the build environment using the oe-init-build-env script
- Add meta-openembedded, meta-virtualization and meta-xilinx layers using bitbake-layers add-layer

Yocto S-DT Build Instructions

```
$ mkdir sources
$ cd sources
$ git clone -b <release-branch> https://git.yoctoproject.org/poky.git
$ git clone -b <release-branch> https://git.openembedded.org/meta-openembedded.git
$ git clone -b <release-branch> https://git.openembedded.org/meta-virtualization.git
$ git clone -b <rel-version> https://github.com/Xilinx/meta-xilinx.git --recurse-submodules
$ source poky/oe-init-build-env
$ bitbake-layers add-layer ./<path-to-layer>/meta-openembedded/meta-oe
$ bitbake-layers add-layer ./<path-to-layer>/meta-openembedded/meta-python
$ bitbake-layers add-layer ./<path-to-layer>/meta-openembedded/meta-filesystems
$ bitbake-layers add-layer ./<path-to-layer>/meta-openembedded/meta-networking
$ bitbake-layers add-layer ./<path-to-layer>/meta-virtualization
$ bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-microblaze
$ bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-xilinx-core
$ bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-xilinx-standalone
$ bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-xilinx-standalone-experimental
```

Lopper Step by Step Transformation

- Clone the lopper and embeddedsw repo
- Export lopper dtc flags
- Transform system device tree to
  - Linux device tree
  - Baremetal device tree
  - FreeRTOS device tree
  - Baremetal Configuration(.h files)
  - Baremetal Drivers
  - Programmable Logic(PL) device tree

Lopper Step by Step Transformation

- Clone the lopper and esw repo

```
$ git clone -b <rel-version> https://github.com/devicetree-org/lopper.git
$ git clone -b <rel-version> https://github.com/Xilinx/embeddedsw.git
$ cd lopper
```

- Variables used in lopper commands

```
LOPPER_OUTPUT_DIR: build-sdt/output/system-top
ESW_REPO_DIR: <path_to_cloned_esw>
SYSTEM_DTFILE: build-sdt/sdtgen/outputs/vek280_noAIE_SegConfig-vek280-sdtws-2024.1/system-top.dts
CONFIG_DTFILE(Linux): build-sdt/conf/dts/xlnx-versal-vek280-revb/cortexa72-versal-linux.dts
CONFIG_DTFILE(APU Baremetal): build-sdt/conf/dts/xlnx-versal-vek280-revb/cortexa72-0-versal-baremetal.dts
CONFIG_DTFILE(APU FreeRTOS): build-sdt/conf/dts/xlnx-versal-vek280-revb/cortexa72-0-versal-freertos.dts
CONFIG_DTFILE(RPU Baremetal): build-sdt/conf/dts/xlnx-versal-vek280-revb/cortexr5-0-versal-baremetal.dts
CONFIG_DTFILE(APU FreeRTOS): build-sdt/conf/dts/xlnx-versal-vek280-revb/cortexr5-0-versal-freertos.dts
CONFIG_DTFILE(MB PMC): build-sdt/conf/dts/xlnx-versal-vek280-revb/microblaze-0-pmc.dts
CONFIG_DTFILE(MB PSM): build-sdt/conf/dts/xlnx-versal-vek280-revb/microblaze-0-psm.dts
SYSTEM_NO_PL_DTFILE: build-sdt/conf/dts/xlnx-versal-vek280-revb/system-top-no-pl.dts
```

Lopper Linux Configuration

- Machine name configuration: Based on board or SoC compatible dt node property

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced -i lops/lop-machine-name.dts ${SYSTEM_DTFILE}
```

- CPU cores Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced -i lops/lop-xilinx-id-cpus.dts ${SYSTEM_DTFILE}
```

- Versal PL overlays Configuration: Lopper generated pl overlay file is found in ${LOPPER_OUTPUT_DIR}/pl.dtsi

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced ${SYSTEM_DTFILE} ${PL_DTFILE} -- xlnx_overlay_dt cortexa72-versal full
```

- Versal Linux Configuration(with PL overlay):

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced -i lops/lop-a72-imux.dts ${SYSTEM_NO_PL_DTFILE} ${CONFIG_DTFILE} -- gen_domain_dts psv_cortexa72_0 linux_dt
```

- Versal Linux Configuration(without PL overlay):

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced -i lops/lop-a72-imux.dts ${SYSTEM_DTFILE} ${CONFIG_DTFILE} -- gen_domain_dts psv_cortexa72_0 linux_dt
```

Lopper Linux Configuration (cont…)

- Machine configuration file output

```
# xlnx-versal-vek280-revb.conf
BBMULTICONFIG += " cortexa72-0-versal-baremetal cortexa72-0-versal-freertos microblaze-0-pmc microblaze-0-psm cortexr5-0-versal-baremetal cortexr5-0-versal-freertos “
TUNEFILE[microblaze-pmc] = "conf/machine/include/xlnx-versal-vek280-revb/microblaze.inc"
TUNEFILE[microblaze-psm] = "conf/machine/include/xlnx-versal-vek280-revb/microblaze.inc“
CONFIG_DTFILE_DIR = "/scratch/sandeep/yocto/2024.1/yp-dev/build-sdt/conf/dts/xlnx-versal-vek280-revb"
CONFIG_DTFILE ?= "${CONFIG_DTFILE_DIR}/cortexa72-versal-linux.dts"
CONFIG_DTFILE[vardepsexclude] += "CONFIG_DTFILE_DIR“
require conf/machine/versal-ai-edge-generic.conf
SYSTEM_DTFILE_DIR = "/scratch/sandeep/yocto/2024.1/yp-dev/build-sdt/sdtgen/outputs/vek280_noAIE_SegConfig-vek280-sdtws-2024.1"
SYSTEM_DTFILE = "${SYSTEM_DTFILE_DIR}/system-top.dts"
SYSTEM_DTFILE[vardepsexclude] += "SYSTEM_DTFILE_DIR“
# Load the dynamic machine features
include conf/machine/include/xlnx-versal-vek280-revb/${BB_CURRENT_MC}-features.conf
LIBXIL_CONFIG = "conf/machine/include/xlnx-versal-vek280-revb/${BB_CURRENT_MC}-libxil.conf"

# Platform Loader and Manager
PLM_DEPENDS = ""
PLM_MCDEPENDS = "mc::microblaze-0-pmc:plm-firmware:do_deploy"
PLM_DEPLOY_DIR = "${MC_TMPDIR_PREFIX}-microblaze-0-pmc/deploy/images/${MACHINE}"

# PSM Firware
PSM_DEPENDS = ""
PSM_MCDEPENDS = "mc::microblaze-0-psm:psm-firmware:do_deploy"
PSM_FIRMWARE_DEPLOY_DIR = "${MC_TMPDIR_PREFIX}-microblaze-0-psm/deploy/images/${MACHINE}“
MC_TMPDIR_PREFIX ??= "${TMPDIR}"
BB_HASHEXCLUDE_COMMON:append = " MC_TMPDIR_PREFIX"
```

Lopper Baremetal or FreeRTOS DTS Configuration

- APU Baremetal Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced -i lops/lop-a72-imux.dts ${SYSTEM_DTFILE} ${CONFIG_DTFILE}
```

- RPU Baremetal Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced -i lops/lop-r5-imux.dts ${SYSTEM_DTFILE} ${CONFIG_DTFILE}
```

Lopper Baremetal or FreeRTOS Driverlist Configuration

- APU Baremetal Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f "${SYSTEM_DTFILE}" --
baremetaldrvlist_xlnx psv_cortexa72_0 "${ESW_REPO_DIR}"
```

- RPU Baremetal Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f "${SYSTEM_DTFILE}" --
baremetaldrvlist_xlnx psv_cortexr5_0 "${ESW_REPO_DIR}"
```

- Outputs:
  - features.conf : Contains list of drivers that are used in the BSP

```
MACHINE_FEATURES = "bram canfd cframe common … uartlite uartpsv usbpsu zdma"
```

  - DRVLISTConfig.cmake : Same list as in features.conf but in a cmake List format
  - ip_drv_map.yaml: Contains mapping of each IP handle against its IP_NAME name and the corresponding embeddedsw driver
  - libxil.conf: Metadata consumed for libxil compilation in yocto mc builds.

```
PACKAGECONFIG[bram] = "${RECIPE_SYSROOT}/usr/lib/libbram.a,,bram,,"
[…]
PACKAGECONFIG[zdma] = "${RECIPE_SYSROOT}/usr/lib/libzdma.a,,zdma,,"
```

Lopper Baremetal or FreeRTOS Configuration file

- Baremetal Configuration file output:

```
# cortexr5-0-versal-baremetal.conf
CONFIG_DTFILE = “/scratch/sandeep/yocto/2024.1/yp-dev/ build-sdt/conf/dts/xlnx-versal-vek280-revb/cortexr5-0-versal-baremetal.dts"
ESW_MACHINE = "psv_cortexr5_0"
DEFAULTTUNE = "cortexr5"
DEF_MC_TMPDIR_PREFIX := "${@d.getVar('MC_TMPDIR_PREFIX', False) or d.getVar('TMPDIR', False)}"
MC_TMPDIR_PREFIX ?= "${DEF_MC_TMPDIR_PREFIX}"
BB_HASHEXCLUDE_COMMON:append = " MC_TMPDIR_PREFIX"
TMPDIR = "${MC_TMPDIR_PREFIX}-cortexr5-0-versal-baremetal"
DISTRO = "xilinx-standalone-nolto"
```

- FreeRTOS Configuration file output:

```
# cortexr5-0-versal-freertos.conf
CONFIG_DTFILE = "/scratch/sandeep/yocto/2024.1/yp-dev/build-sdt/conf/dts/xlnx-versal-vek280-revb/cortexr5-0-versal-freertos.dts"
ESW_MACHINE = "psv_cortexr5_0"
DEFAULTTUNE = "cortexr5"
DEF_MC_TMPDIR_PREFIX := "${@d.getVar('MC_TMPDIR_PREFIX', False) or d.getVar('TMPDIR', False)}"
MC_TMPDIR_PREFIX ?= "${DEF_MC_TMPDIR_PREFIX}"
BB_HASHEXCLUDE_COMMON:append = " MC_TMPDIR_PREFIX"
TMPDIR = "${MC_TMPDIR_PREFIX}-cortexr5-0-versal-freertos"
DISTRO = "xilinx-freertos"
```

Lopper MicroBlaze PMC Configuration

- MicroBlaze PMC DTS Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced
${SYSTEM_DTFILE} ${CONFIG_DTFILE}
```

- MicroBlaze PMC Driverlist Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f "${SYSTEM_DTFILE}" --
baremetaldrvlist_xlnx psv_pmc_0 "${ESW_REPO_DIR}"
```

Lopper MicroBlaze PSM Configuration

- MicroBlaze PSM DTS Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced
${SYSTEM_DTFILE} ${CONFIG_DTFILE}
```

- MicroBlaze PSM Driverlist Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f "${SYSTEM_DTFILE}" --
baremetaldrvlist_xlnx psv_psm_0 "${ESW_REPO_DIR}"
```

Lopper MicroBlaze Tune Configuration

- MicroBlaze Tunes Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f --enhanced -i lops/lop-microblaze-yocto.dts ${SYSTEM_DTFILE}
```

- Outputs:

```
# microblaze-0-pmc.conf
CONFIG_DTFILE = "/scratch/sandeep/yocto/2024.1/yp-dev/build-sdt/conf/dts/xlnx-versal-vek280-revb/microblaze-0-pmc.dts"
ESW_MACHINE = "psv_pmc_0"
DEFAULTTUNE = "microblaze-pmc"
DEF_MC_TMPDIR_PREFIX := "${@d.getVar('MC_TMPDIR_PREFIX', False) or d.getVar('TMPDIR', False)}"
MC_TMPDIR_PREFIX ?= "${DEF_MC_TMPDIR_PREFIX}"
BB_HASHEXCLUDE_COMMON:append = " MC_TMPDIR_PREFIX"
TMPDIR = "${MC_TMPDIR_PREFIX}-microblaze-0-pmc"
DISTRO = "xilinx-standalone"
TARGET_CFLAGS += "-DVERSAL_PLM=1"
# microblaze-0-psm.conf
CONFIG_DTFILE = "/scratch/sandeep/yocto/2024.1/yp-dev/build-sdt/conf/dts/xlnx-versal-vek280-revb/microblaze-0-psm.dts"
ESW_MACHINE = "psv_psm_0"
DEFAULTTUNE = "microblaze-psm"
DEF_MC_TMPDIR_PREFIX := "${@d.getVar('MC_TMPDIR_PREFIX', False) or d.getVar('TMPDIR', False)}"
MC_TMPDIR_PREFIX ?= "${DEF_MC_TMPDIR_PREFIX}"
BB_HASHEXCLUDE_COMMON:append = " MC_TMPDIR_PREFIX"
TMPDIR = "${MC_TMPDIR_PREFIX}-microblaze-0-psm"
DISTRO = "xilinx-standalone-nolto"
TARGET_CFLAGS += "-DVERSAL_psm=1"
```

Lopper Params Configuration

- APU Baremetal Params Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f "${SYSTEM_DTFILE}" --
baremetal_xparameters_xlnx psv_cortexa72_0 "${ESW_REPO_DIR}"
```

- RPU Baremetal Params Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f "${SYSTEM_DTFILE}" --
baremetal_xparameters_xlnx psv_cortexr5_0 "${ESW_REPO_DIR}"
```

- Microblaze Params Configuration:

```
$ LOPPER_DTC_FLAGS="-b 0 -@" ./lopper.py -O ${LOPPER_OUTPUT_DIR} -f "${SYSTEM_DTFILE}" --
baremetal_xparameters_xlnx psv_pmc_0 "${ESW_REPO_DIR}/scripts/pyesw/.repo.yaml"
```

- Outputs:

`xparameters.h`

Generating YP Config using S-DT for AMD Versal AI-Edge

- User can use lopper tool and transform system device tree files to operating environment standard device tree files
- Alternatively using meta-xilinx yocto project layers we can generate multiconfig files using dt-processor.sh script available from system device tree

dt-processor.sh method

- Follow S-DT build instructions
- Build the setup SDK
- Install the setup SDK
- Source dt-process.sh to generates dts, machine and multiconfig files
- Build the multiconfig target images
- Emulate the images using QEMU

dt-processor.sh method (cont…)

```
$ bitbake meta-xilinx-setup
$ .${TMPDIR}/tmp/deploy/sdk/x86_64-xilinx-nativesdk-prestep-2023.2....sh -d prestep -y
$ . prestep/environment-setup-x86_64-petalinux-linux
$ ./prestep/dt-processor.sh -c conf/ -s sdt/vek28080_dts/system-top.dts -l
conf/local.conf
$ time bitbake core-image-minimal
```

Generated DTS, Machine and Multiconfig

```
├── bblayers.conf
├── dts
│   └── xlnx-versal-vek280-revb
│       ├── cortexa72-0-versal-baremetal.dts
│       ├── cortexa72-0-versal-freertos.dts
│       ├── cortexa72-1-versal-baremetal.dts
│       ├── cortexa72-1-versal-freertos.dts
│       ├── cortexa72-versal-linux.dts
│       ├── cortexr5-0-versal-baremetal.dts
│       ├── cortexr5-0-versal-freertos.dts
│       ├── cortexr5-1-versal-baremetal.dts
│       ├── cortexr5-1-versal-freertos.dts
│       ├── microblaze-0-pmc.dts
│       ├── microblaze-0-psm.dts
│       ├── pl-overlay-full
│       │   └── pl.dtsi
│       └── system-top-no-pl.dts
├── local.conf
├── machine
│   ├── include
│   │   └── xlnx-versal-vek280-revb
│   │       ├── cortexa72-0-versal-baremetal-features.conf
│   │       ├── cortexa72-0-versal-baremetal-libxil.conf
│   │       ├── cortexa72-0-versal-freertos-features.conf
│   │       ├── cortexa72-0-versal-freertos-libxil.conf
│   │       ├── cortexa72-1-versal-baremetal-features.conf
│   │       ├── cortexa72-1-versal-baremetal-libxil.conf
│   │       ├── cortexa72-1-versal-freertos-features.conf
│   │       ├── cortexa72-1-versal-freertos-libxil.conf
│   │       ├── cortexr5-0-versal-baremetal-features.conf
│   │       ├── cortexr5-0-versal-baremetal-libxil.conf
│   │       ├── cortexr5-0-versal-freertos-features.conf
│   │       ├── cortexr5-0-versal-freertos-libxil.conf
│   │       ├── cortexr5-1-versal-baremetal-features.conf
│   │       ├── cortexr5-1-versal-baremetal-libxil.conf
│   │       ├── cortexr5-1-versal-freertos-features.conf
│   │       ├── cortexr5-1-versal-freertos-libxil.conf
│   │       ├── microblaze-0-pmc-features.conf
│   │       ├── microblaze-0-pmc-libxil.conf
│   │       ├── microblaze-0-psm-features.conf
│   │       ├── microblaze-0-psm-libxil.conf
│   │       └── microblaze.inc
│   └── xlnx-versal-vek280-revb.conf
└── multiconfig
    ├── cortexa72-0-versal-baremetal.conf
    ├── cortexa72-0-versal-freertos.conf
    ├── cortexa72-1-versal-baremetal.conf
    ├── cortexa72-1-versal-freertos.conf
    ├── cortexr5-0-versal-baremetal.conf
    ├── cortexr5-0-versal-freertos.conf
    ├── cortexr5-1-versal-baremetal.conf
    ├── cortexr5-1-versal-freertos.conf
    ├── default.conf
    ├── microblaze-0-pmc.conf
    └── microblaze-0-psm.conf
```

Gen-Machine-Conf

- Follow the Yocto S-DT build instructions
- Export gen-machineconf tool
- Using S-DT as input run gen-machinconf tool to create dts, machine and multiconfig files
- Build the multiconfig target images
- Emulate the images using QEMU
- See Gen-Machine-Conf talk from YP Summit
  - eLinux landing page - https://elinux.org/YPS_2023.11_Presentations
  - Slides - https://summit.yoctoproject.org/media/yocto-project-summit-2023-11/submissions/HGWFRT/resources/Yocto_Project_Summit_2023.11_G_AQaMrZD.pdf
  - Video - https://youtu.be/UQQCP9jPzps?si=EI0Ay65BmoAweoGy

Gen-Machine-Conf (cont…)

- Without S-DT PL overlay

```
$ gen-machineconf --hw-description <path_to_sdtgen_output_directory> -c <conf> -l
conf/local.conf
```

- ZynqMP Full or Versal Segmented Configuration

```
$ gen-machineconf --hw-description <path_to_sdtgen_output_directory> -c <conf> -l
conf/local.conf –g full
```

- ZynqMP or Versal DFx

```
$ gen-machineconf --hw-description <path_to_sdtgen_output_directory> -c <conf> -l
conf/local.conf –g dfx
```

What Next in Scarthgap

What Next in Scarthgap

- Remove dt-processor.sh support from meta-xilinx layer
- Gen-Machine-Conf tool to support modular approach
- Detect machine type dynamically using S-DT
- Bitbake Class to support packaging and deploying multiconfig firmware elf or bin(openamp) to linux rootfs image
- Zynq-7000 and MicroBlaze* S-DT and Lopper support
  - Targeting fall 2024

QUESTIONS


AMD